Annex

A.1 Results Visualization with Saliency Maps

Due to the lack of consistency in the results obtained with this methodology we decided to exclude it from the main project. The results are, nevertheless, interesting as it shows sometimes possible explications for the decisions of the algorithms.

A.1.1 Original uncorrupted images

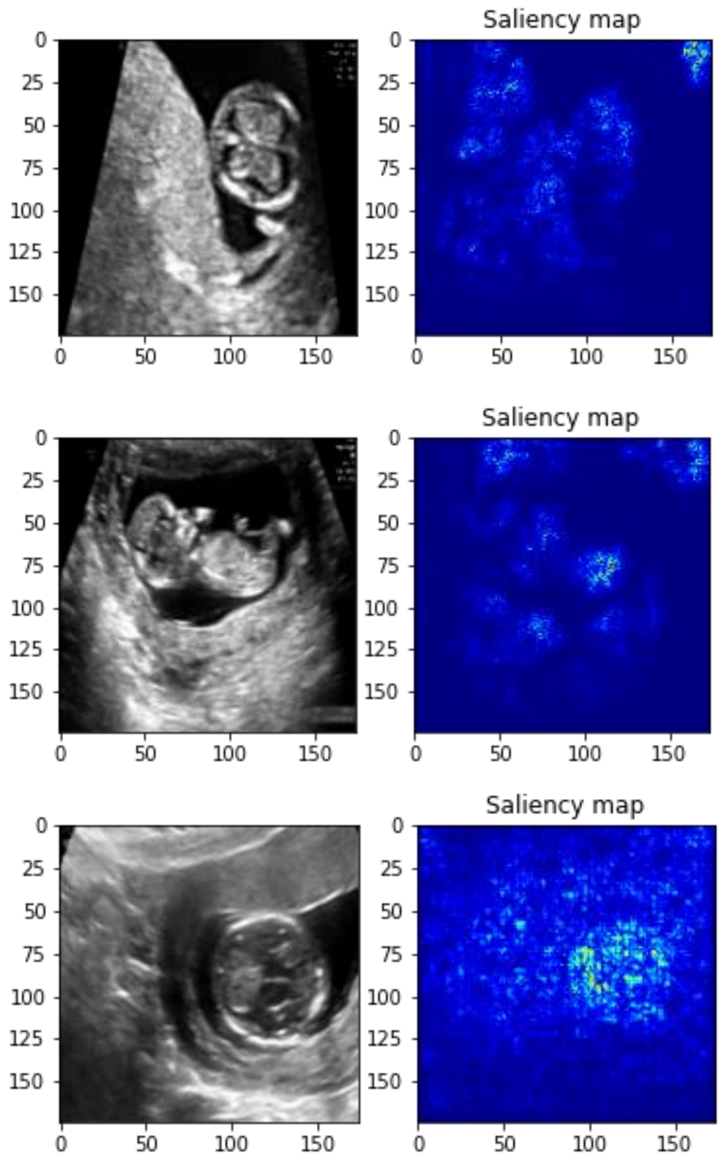

Figure 35. Saliency maps of correctly classified Original (unmodified) images. The CNN seems to focus on the white structed shapes including the fetus. (Source: own)

A.1.2 Scaled up images

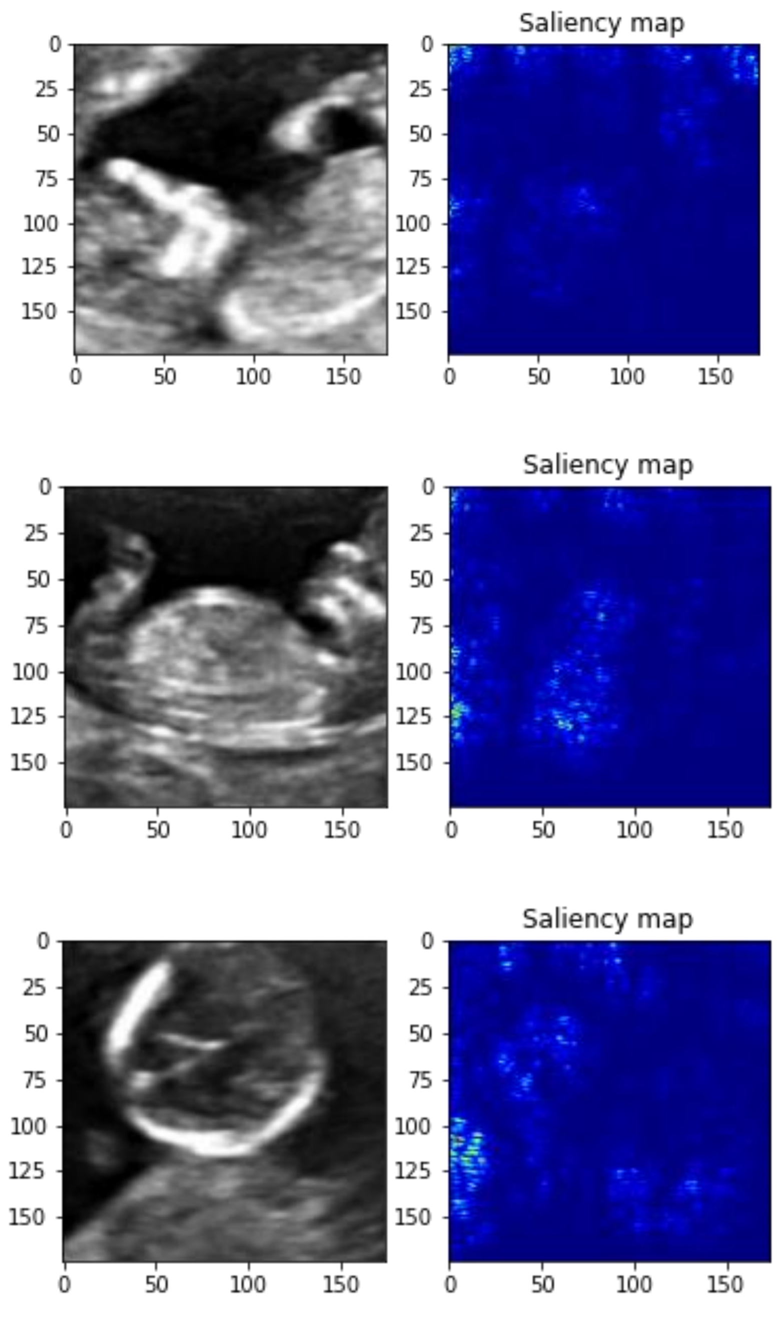

Figure 36. Saliency maps of correctly classified scaled up images. The model seems to focus not only on the fetus but also on the edges of the images where the black shapes appear drastically cut due to the corruption. (Source: own)

A.1.3 Blurred Images

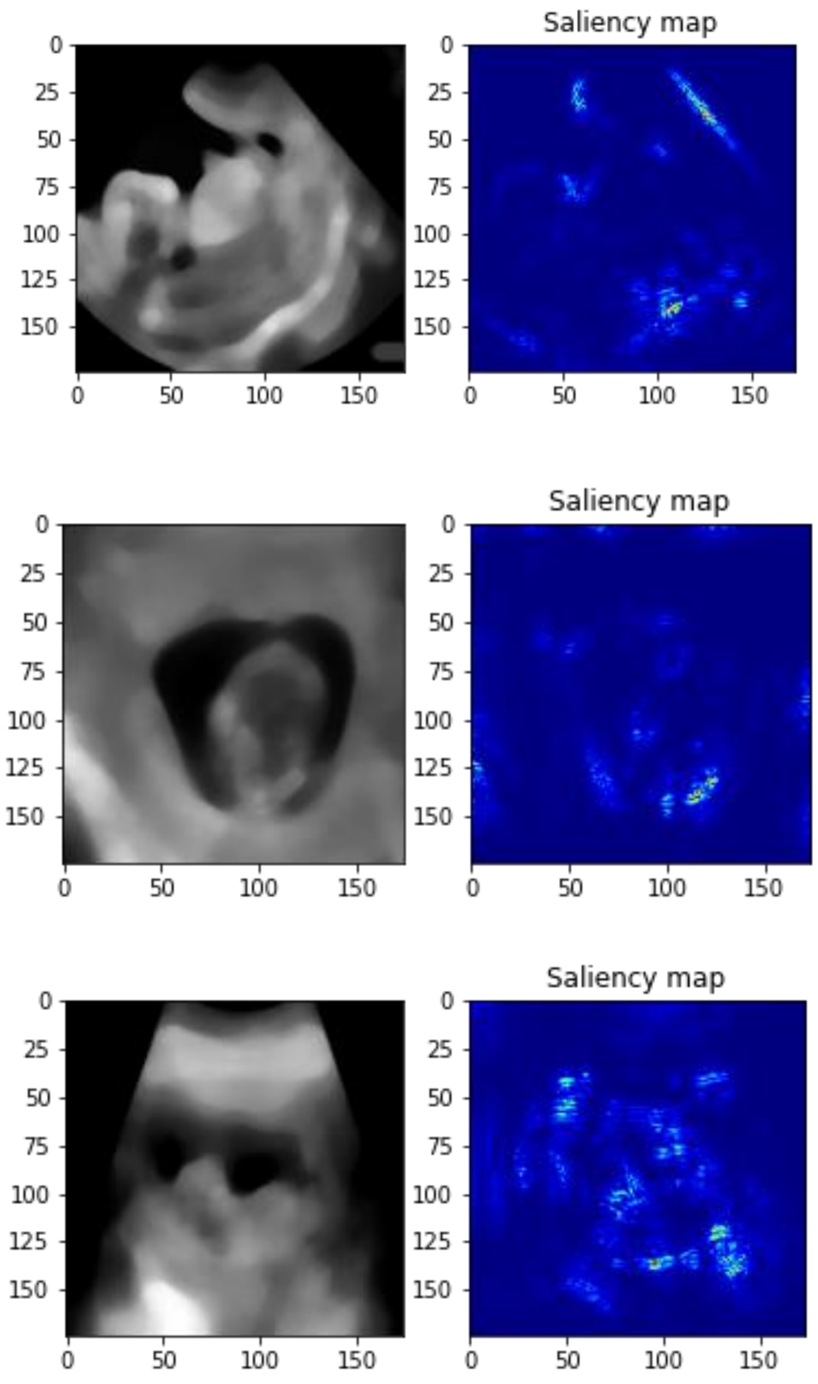

Figure 37. Saliency maps of correctly classified Blurred images. The model seems to especially focus on edges. (Source: own)

A.1.4 Noised Images

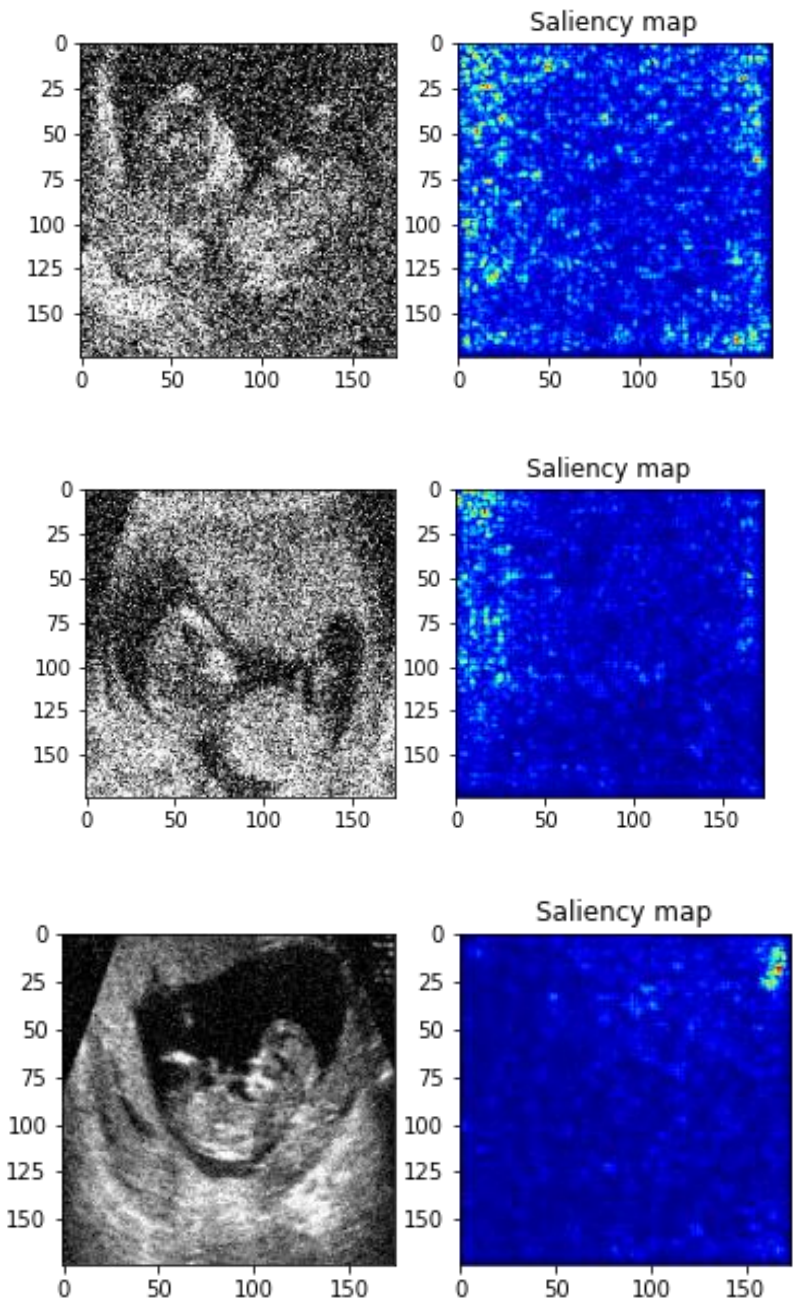

Figure 38. Saliency maps of correctly classified Noised images. The models seem to focus on affected pixels where the contrast between the intensity of the original image and the noise are greater. One interesting result is the last image where the model recognizes some partially visible text (due to not fully removal in the cleaning process) and identifies it as noise. (Source: own)

A.2 Results Comparison on different plots

Again, this section has not been included in the main report because all data has been presented in other formats. However, we believe that these additional plots can help on further understanding the results.

We will present them with a description to help understand the context.

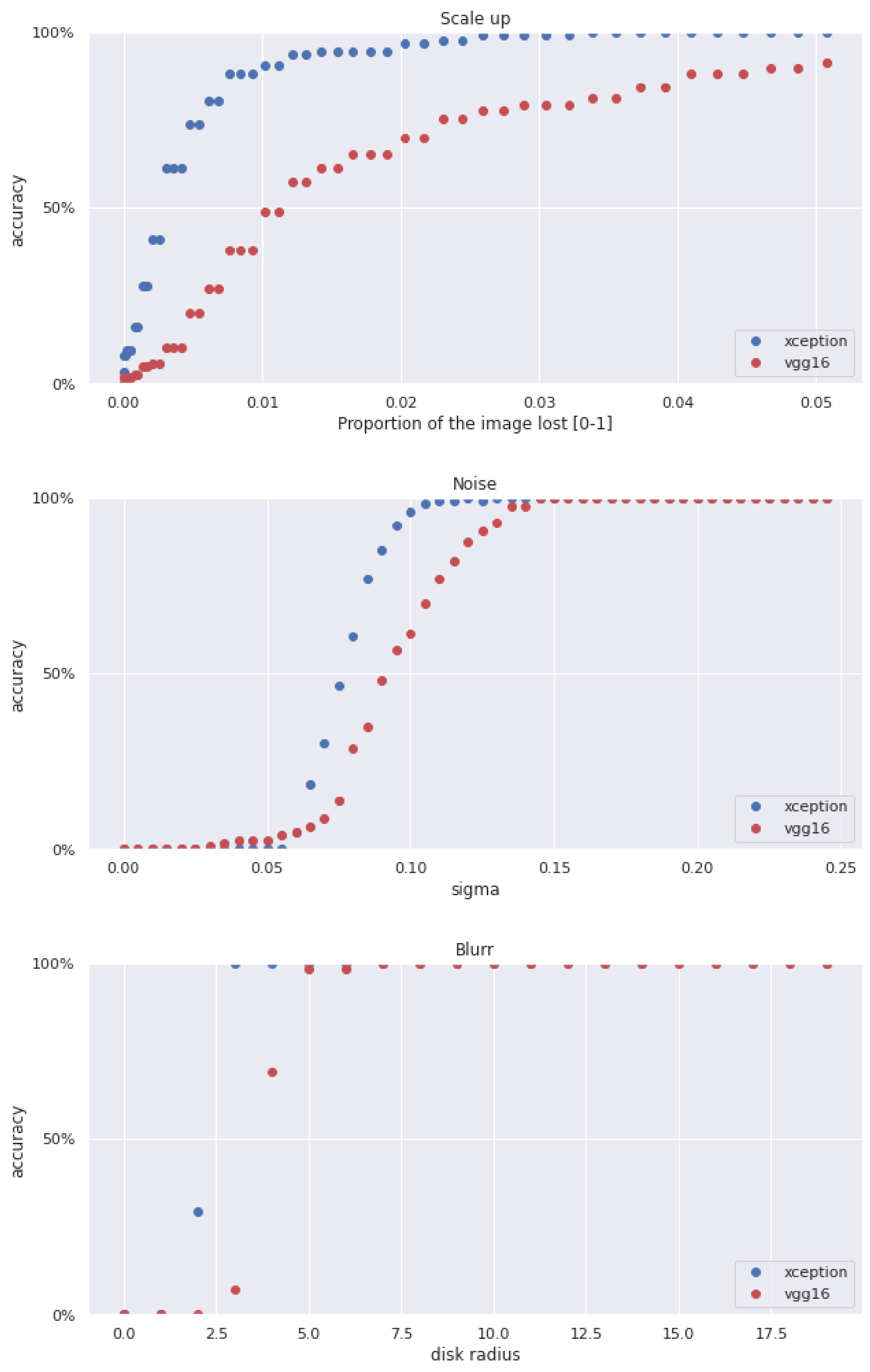

Figure 39. Result Comparison of Accuracy by corruption intensity. In this case the limit value between corruption and correct images is not set and so, all images are labeled as correct. This is reflected by the 0% accuracy at the beginning as the model classifies them as correct. The behavior seems opposite as the one we saw in Figure 21. (Source: own)

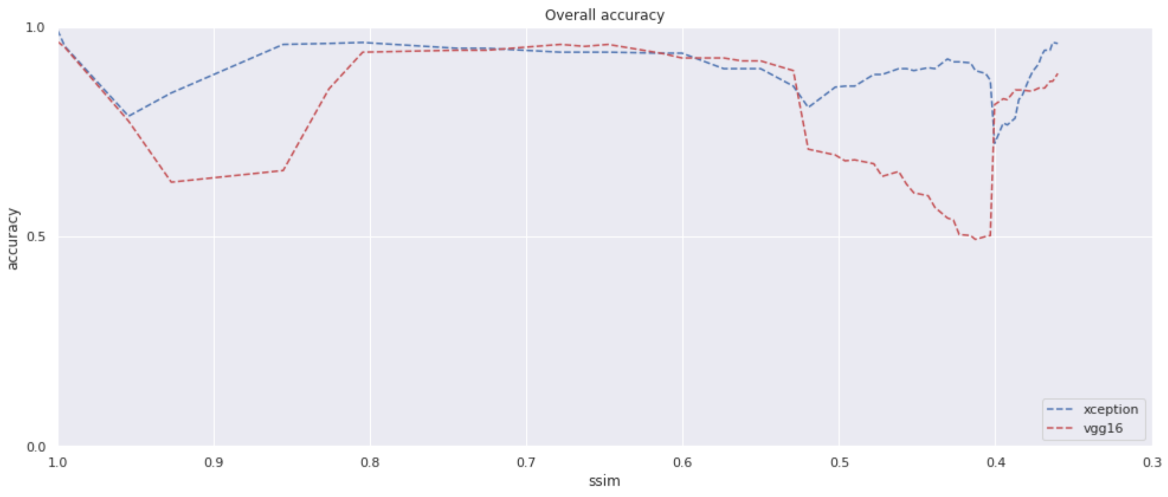

Figure 40. Result comparison of overall average accuracy by structural similarity. The Xception model outperforms de VGG16. The curve drops on multiple zones due to the reason that each corruption has a drop on a different ssim. When using the average, the result is softened as is the average between the “dropping” corruption and the other two. (Source: own)

A.3 Code

Image Corruption Generation

```
#---------------------MOUNT DRIVE----------------------
from google.colab import drive
drive.mount('/content/drive')
#---------------------IMPORT----------------------
import os
import skimage as sk
import skimage.io
import sklearn as skl
from sklearn.model_selection import train_test_split
import numpy as np
import cv2

from skimage.morphology import disk
from skimage.filters import median
from skimage.util import random_noise
import random
#---------------------GENERATE IMAGES----------------------
#Get the the Image folders
directori1='/content/drive/My Drive/TFM Lluis Cierco - DeepLearning
/Agrupades'
y1=os.listdir(directori1)

images_orig=[]
y_orig=[]

images_mod=[]
y_mod=[]
y_type=[]
#Transform each image and store it in a list
#Original images are rotated to keep same amount of data as distors
ionate

width=224 #227
height=224 #337

for image in y1:
  im_orig=sk.io.imread(os.path.join(directori1,image), as_gray=True
)
  im_orig=cv2.resize(im_orig, (width-50,height-50))

  #VGG16 needs color images
  im_orig=cv2.merge((im_orig,im_orig,im_orig))

  images_orig.append(im_orig)

  if image.split("_")[0]=="BPD":
    y_type.append(0)
  elif image.split("_")[0]=="CLR":
    y_type.append(1)
```

Model Creation and Training

```
#---------------------IMPORT----------------------
from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.xception import Xception

#---------------------PREPARE DATA FOR CNN----------------------
#Merge lists of images and merge lists of labels
all_images = images_orig + images_mod
all_y = y_orig+y_mod


#Split train and test images
x_train, x_test, y_train, y_test = train_test_split(all_images,all_
y)

#Reshape into np array
x_train=np.array(x_train).reshape(np.shape(x_train)[0],width-
50,height-50,3)
x_test=np.array(x_test).reshape(np.shape(x_test)[0],width-
50,height-50,3)

# Code classes on shape (4,) arrays
y_train = utils.to_categorical(y_train)
y_test= utils.to_categorical(y_test)
#---------------------MODEL CNN - DEEP LEARNING--------------------
-
# Create model:
def baseline_model():
    model=Sequential()
    model.add(Xception(include_top=False, input_shape=(width-
50,height-50,3)))
    #model.add(VGG16(include_top=False, input_shape=(width-
50,height-50,3)))
    model.add(Flatten())
    model.add(Dense(1028, activation='relu'))
    model.add(Dense(4, activation='softmax'))
    return model

# Define model
model = baseline_model()
model.compile(optimizer='adam', loss='categorical_crossentropy', me
trics=['accuracy'])
```

Full Range of Test Images Generation

```
#CREATE CORRUPTIONS
import tensorflow
resultat=[]
resultat_2=[]
predictions=[]
predictions_2=[]
grau_dist=[]

distorsions=['Crop','Noise', 'Blurr']
for distorsio in distorsions:
  intensity=-0.05
  print('Complete')
  RANG=50
  if distorsio=='Blurr':
    RANG=20
  for proba in range(RANG):
    n=1
    intensity=intensity+0.05
    img_distorsionades=[]
    grau_dist_ind=0.0
    img_distorsionades.append(x_test[0])
    img_distorsionades.append(x_test[1])
    img_distorsionades.append(x_test[2])
    img_distorsionades.append(x_test[3])
    y_dist=[0,1,2,3]
    if distorsio=='Crop':
      intensity = intensity + 0.06
    for i in range(len(y_test)):
        img = x_test[i][:,:,0]

        if y_test[i][0]==1:

          if distorsio=='Crop':
            img=cv2.resize(img, (width-50,height-50))
            width_crop_a=intensity*4
            width_crop_b=intensity*4
            height_crop_a=intensity*4
            height_crop_b=intensity*4

            im_crop=sk.util.crop(img, ((width_crop_a,  width_crop_b
),(height_crop_a,height_crop_b)))
            im_crop=cv2.resize(im_crop, (width-50,height-50))
            im_crop=cv2.merge((im_crop,im_crop,im_crop))

            img_distorsionades.append(im_crop)
            y_dist.append(1)

          elif distorsio=='Blurr':
            sel25 = disk((intensity/0.05))
```

```
 img_med25x25 = median(img, sel25)
           img_med25x25=cv2.merge((img_med25x25,img_med25x25,img_med25x25))
           img_distorsionades.append(img_med25x25)
           y_dist.append(2)

         else:
           sigma = intensity/20.0
           noisy_image_normal = random_noise(img, var=(sigma)**2.0)
           noisy_image_normal=cv2.merge((noisy_image_normal,noisy_image_normal,noisy_im
age_normal))
           img_distorsionades.append(noisy_image_normal)
           y_dist.append(3)

         ssim = sk.metrics.structural_similarity(x_test[i], img_distorsionades[-
1], multichannel=True)
         grau_dist_ind= grau_dist_ind + ssim

    y_dist = tensorflow.keras.utils.to_categorical(y_dist)
    img_distorsionades=np.array(img_distorsionades).reshape(np.shape(img_distorsionades)
[0],width-50,height-50,3)

    resultat_dist = model.evaluate(img_distorsionades[4:], y_dist[4:], verbose=0)
    prediction = (model.predict(img_distorsionades[4:]) >0.5).astype(int)

    resultat_dist_2 = model_2.evaluate(img_distorsionades[4:], y_dist[4:], verbose=0)
    prediction_2 = (model_2.predict(img_distorsionades[4:]) >0.5).astype(int)

    resultat.append(resultat_dist)
    resultat_2.append(resultat_dist_2)

    predictions.append(prediction)
    predictions_2.append(prediction_2)

    grau_dist.append(grau_dist_ind/len(img_distorsionades[4:]))
```

Saliency Maps

```python
# Saliency Maps code
# =============================================
from vis.visualization import visualize_saliency
from vis.utils import utils
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import activations



# Find the index of the to be visualized layer above
#layer_index = utils.get_layer()find_layer_idx(model, 'vgg16')
layer_index=-1
# Swap softmax with linear
model.layers[-1].activation = activations.linear
#model = utils.apply_modifications(model)

# Numbers to visualize
indices_to_visualize = [2, 61, 111, 212, 312, 433, 521 ]

# Visualize
for index_to_visualize in indices_to_visualize:
  # Get input
  input_image = x_test[index_to_visualize]
  # Class object
  classes = {
    0: 'Original',
    1: 'Crop',
    2: 'Blurr',
    3: 'Noise',
  }
  input_class = np.argmax(y_test[index_to_visualize])
  input_class_name = classes[input_class]
  # Matplotlib preparations
  fig, axes = plt.subplots(1, 2)
  # Generate visualization
  visualization = visualize_saliency(model, layer_index, filter_ind
ices=input_class, seed_input=input_image)
  axes[0].imshow(input_image)
  axes[0].set_title('')
  axes[1].imshow(visualization)
  axes[1].set_title('Saliency map')
  fig.suptitle(f'Class = {input_class_name}')
  plt.show()
```

ETSEIB